

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE APPLICATION OF: Yutaka OTA

GAU:

SERIAL NO: New Application

EXAMINER:

FILED: Herewith

FOR: COMPILER, METHOD OF COMPILING AND PROGRAM DEVELOPMENT TOOL

REQUEST FOR PRIORITY

COMMISSIONER FOR PATENTS
ALEXANDRIA, VIRGINIA 22313

SIR:

☐ Full benefit of the filing date of U.S. Application Serial Number _____, filed _____, is claimed pursuant to the provisions of **35 U.S.C. §120**.

☐ Full benefit of the filing date(s) of U.S. Provisional Application(s) is claimed pursuant to the provisions of **35 U.S.C. §119(e)**:
Application No. _____ **Date Filed** _____

☒ Applicants claim any right to priority from any earlier filed applications to which they may be entitled pursuant to the provisions of **35 U.S.C. §119**, as noted below.

In the matter of the above-identified application for patent, notice is hereby given that the applicants claim as priority:

COUNTRY

Japan

APPLICATION NUMBER

2003-085848

MONTH/DAY/YEAR

March 26, 2003

Certified copies of the corresponding Convention Application(s)

☒ are submitted herewith

☐ will be submitted prior to payment of the Final Fee

☐ were filed in prior application Serial No. _____ filed _____

☐ were submitted to the International Bureau in PCT Application Number _____

Receipt of the certified copies by the International Bureau in a timely manner under PCT Rule 17.1(a) has been acknowledged as evidenced by the attached PCT/IB/304.

☐ (A) Application Serial No.(s) were filed in prior application Serial No. _____ filed _____; and

☐ (B) Application Serial No.(s) _____

☐ are submitted herewith

☐ will be submitted prior to payment of the Final Fee

Respectfully Submitted,

OBLON, SPIVAK, McCLELLAND,
MAIER & NEUSTADT, P.C.



Marvin J. Spivak

Registration No. 24,913

C. Irvin McClelland
Registration Number 21,124

Customer Number

22850

Tel. (703) 413-3000
Fax. (703) 413-2220
(OSMMN 05/03)

日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出 願 年 月 日

Date of Application:

2003年 3月26日

出 願 番 号

Application Number:

特願2003-085848

[ST.10/C]:

[JP 2003-085848]

出 願 人

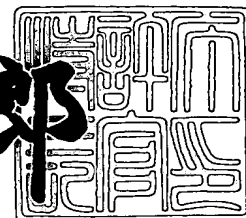
Applicant(s):

株式会社東芝

2003年 4月18日

特 許 庁 長 官
Commissioner,
Japan Patent Office

太田 信一郎



出証番号 出証特2003-3028153

【書類名】 特許願

【整理番号】 ACB02Z003

【提出日】 平成15年 3月26日

【あて先】 特許庁長官殿

【国際特許分類】 G06F 9/44

【発明の名称】 コンパイラ、コンパイル方法、及びプログラム開発ツール

【請求項の数】 15

【発明者】

 【住所又は居所】 神奈川県川崎市幸区小向東芝町1番地 株式会社東芝
 マイクロエレクトロニクスセンター内

 【氏名】 太田 裕

【特許出願人】

 【識別番号】 000003078

 【氏名又は名称】 株式会社 東芝

【代理人】

 【識別番号】 100083806

 【弁理士】

 【氏名又は名称】 三好 秀和

 【電話番号】 03-3504-3075

【選任した代理人】

 【識別番号】 100068342

 【弁理士】

 【氏名又は名称】 三好 保男

【選任した代理人】

 【識別番号】 100100712

 【弁理士】

 【氏名又は名称】 岩▲崎▼ 幸邦

【選任した代理人】

【識別番号】 100100929

【弁理士】

【氏名又は名称】 川又 澄雄

【選任した代理人】

【識別番号】 100108707

【弁理士】

【氏名又は名称】 中村 友之

【選任した代理人】

【識別番号】 100095500

【弁理士】

【氏名又は名称】 伊藤 正和

【選任した代理人】

【識別番号】 100101247

【弁理士】

【氏名又は名称】 高橋 俊一

【選任した代理人】

【識別番号】 100098327

【弁理士】

【氏名又は名称】 高松 俊雄

【手数料の表示】

【予納台帳番号】 001982

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 コンパイラ、コンパイル方法、及びプログラム開発ツール

【特許請求の範囲】

【請求項 1】 入力したソースプログラムからオブジェクトコードを生成するコンパイラであって、

前記ソースプログラム内に記述された命令について文法規則への適合性を解析し、前記命令の組み合わせが組込関数及び該組込関数の動作内容を定義しているか否かを解析する構文解析部と、

前記構文解析部で解析された前記組込関数及び該組込関数の動作内容の定義を記憶する組込関数定義記憶部と、

前記構文解析部の処理結果に基づいて、前記ソースプログラムから機械語を生成するコード生成部と、

前記コード生成部により生成された前記機械語が前記組込関数定義記憶部に記憶されている前記組込関数の動作内容と一致する場合、当該機械語を前記組込関数の動作内容に応じた機械語に最適化するコード最適化部と

を備えることを特徴とするコンパイラ。

【請求項 2】 前記ソースプログラム内に記述された命令をトークンに分割する字句解析部を備え、

前記構文解析部は、前記トークンについて文法規則への適合性を解析し、前記トークンの組み合わせが前記組込関数及び該組込関数の動作内容を定義しているか否かを解析することを特徴とする請求項 1 に記載のコンパイラ。

【請求項 3】 前記字句解析部は前記組込関数及びその動作内容の定義を外部ファイルから入力することを特徴とする請求項 1 又は請求項 2 に記載のコンパイラ。

【請求項 4】 前記組込関数の定義には、仮引数型と識別子名とが含まれることを特徴とする請求項 1 乃至請求項 3 のいずれか 1 項に記載のコンパイラ。

【請求項 5】 前記組込関数定義記憶部には、1 つの組込関数に対して複数種類の動作内容を定義できることを特徴とする請求項 1 乃至請求項 4 のいずれか 1 項に記載のコンパイラ。

【請求項 6】 前記組込関数の動作内容には、機械語列が含まれることを特徴とする請求項 1 乃至請求項 5 のいずれか 1 項に記載のコンパイラ。

【請求項 7】 前記組込関数及びその動作内容の定義は、ハードウェア記述言語で記述されていることを特徴とする請求項 1 乃至請求項 6 のいずれか 1 項に記載のコンパイラ。

【請求項 8】 入力したソースプログラムからオブジェクトコードを生成するコンパイラによるコンパイル方法であって、

前記ソースプログラム内に記述された命令の文法規則への適合性を解析し、前記命令の組み合わせが組込関数及び該組込関数の動作内容を定義しているか否かを構文解析部が解析する段階と、

前記構文解析部により解析された前記組込関数及び該組込関数の動作内容の定義を組込関数定義記憶部に記憶する段階と、

前記構文解析部の処理結果に基づいて、コード生成部が前記ソースプログラムから機械語を生成する段階と、

前記コード生成部により生成された前記機械語が前記組込関数定義記憶部に記憶された前記組込関数の動作内容と一致する場合、コード最適化部が当該機械語を前記組込関数の動作内容に応じた機械語に最適化する段階と

を含むことを特徴とするコンパイル方法。

【請求項 9】 前記ソースプログラム内に記述された命令を字句解析部がトークンに分割する段階を含み、

前記構文解析部による解析段階において、前記トークンについて文法規則への適合性を解析し、前記トークンの組み合わせが前記組込関数及び該組込関数の動作内容を定義しているか否かを解析することを特徴とする請求項 8 に記載のコンパイル方法。

【請求項 10】 前記字句解析部は前記組込関数及びその動作内容の定義を外部ファイルから入力することを特徴とする請求項 8 又は請求項 9 に記載のコンパイル方法。

【請求項 11】 前記構文解析部が解析し、前記組込関数定義記憶部に記憶する前記組込関数の定義には、仮引数型と識別子名とが含まれることを特徴とす

る請求項 8 乃至請求項 1 0 のいずれか 1 項に記載のコンパイル方法。

【請求項 1 2】 前記組込関数定義記憶部には、1 つの組込関数に対して複数種類の動作内容を定義できることを特徴とする請求項 8 乃至請求項 1 1 のいずれか 1 項に記載のコンパイル方法。

【請求項 1 3】 前記構文解析部が解析し、前記組込関数定義記憶部に記憶する前記組込関数の動作内容には、機械語列が含まれることを特徴とする請求項 8 乃至請求項 1 2 のいずれか 1 項に記載のコンパイル方法。

【請求項 1 4】 前記構文解析部は、ハードウェア記述言語で記述されている前記組込関数及びその動作内容の定義を解析することを特徴とする請求項 8 乃至請求項 1 3 のいずれか 1 項に記載のコンパイル方法。

【請求項 1 5】 ユーザ定義のハードウェアを導入したプロセッサのためのアプリケーションプログラムを設計するためのプログラム開発ツールであって、

前記アプリケーションプログラム内に記述された命令をトークンに分割する字句解析部と、前記トークンについて文法規則への適合性を解析し、ユーザ定義命令のハードウェア定義を入力して組込関数及び該組込関数の動作内容の定義に変換する構文解析部と、前記構文解析部で変換された前記組込関数及び該組込関数の動作内容の定義を記憶する組込関数定義記憶部と、前記字句解析部及び前記構文解析部の処理結果に基づいて、前記アプリケーションプログラムから機械語を生成するコード生成部と、前記コード生成部により生成された前記機械語が前記組込関数定義記憶部に記憶されている前記組込関数の動作内容と一致する場合、当該機械語を前記組込関数の動作内容に応じた機械語に最適化するコード最適化部とを含むコンパイラと、

前記コンパイラによりコンパイルされた前記アプリケーションプログラムをデバッグするシミュレータと

を備えることを特徴とするプログラム開発ツール。

【発明の詳細な説明】

【0 0 0 1】

【発明の属する技術分野】

プログラミング言語により記述されたソースプログラムを、機械語によるオブ

ジェクトコードに変換（コンパイル）するコンパイラ、コンパイル方法、及び該コンパイラを備えたプログラム開発ツールに係り、特に、ユーザによる拡張仕様を定義した関数を組み込む技術に関する。

【0002】

【従来の技術】

一般に、ソースプログラムを機械語であるオブジェクトコードに変換するコンパイラは、プロセッサ・アーキテクチャごとに用意され、関数の処理手順もコンパイラ内部に組み込まれていて、ユーザが独自に変更できない。従って、ユーザが仕様（プロセッサ・アーキテクチャおよび命令セット）を拡張可能なプロセッサでアプリケーションプログラム等を動作させるためには、ユーザ仕様に対応したコンパイラが必要となる。

【0003】

そこで、本出願人は、組込関数（intrinsic function）をユーザの拡張仕様に応じてカスタマイズできるコンパイラを提案している（例えば、特許文献1参照）。

【0004】

【特許文献1】

特開2002-24029号公報

【0005】

【発明が解決しようとする課題】

しかしながら、特許文献1に記載のコンパイラでは、ユーザが定義した組込関数を用いた処理動作と同等の動作を行う命令文を、ソースプログラム中にプログラミング言語で記述したとしても、その命令文をユーザ定義の組込関数に応じた機械語に変換することができなかった。

【0006】

例えば、ソースレジスタに"10"を加算して、"__imm"でマスクした値をデスティネーションレジスタに格納する処理として、図19（a）に示すような組込関数"uci"（組込関数F91）をソースプログラム中に宣言した場合、コンパイラは、ソースプログラム中で組込関数"uci"を明示的に呼び出す命令文T91から

、ユーザの定義に応じた命令語"uci"を用いた機械語（図 1 9 （b）における機械語M 9 1）へ変換することが可能である。

【0 0 0 7】

ところが、命令文T 9 1 が組込関数"uci"を用いて実行する処理動作（ソースレジスタに"10"を加算して、"_imm"でマスクした値をデスティネーションレジスタに格納する処理動作）と同等の処理動作を実行する命令文T 9 2 がソースプログラム中に記述されていたとしても、コンパイラは命令文T 9 2 をユーザ定義の命令語"uci"を用いた機械語へ変換することができず、命令語"add"を用いた従来の機械語（図 1 9.（b）における機械語M 9 2）を生成してしまう。命令文T 9 2 をユーザ定義の命令語"uci"を用いた機械語へ変換させるためには、ユーザは、命令文T 9 2 の記述を、組込関数"uci"を明示的に呼び出す命令文T 9 1 に書き改める必要がある。

【0 0 0 8】

このように、ユーザは、ソースプログラムを記述する時に、ユーザが定義した組込関数について熟知していなければならなかったり、プロセッサの仕様を拡張した時に、ソースプログラム中の記述の中から該当する動作部分をユーザが定義した組込関数に書き換える必要があったりして、プロセッサあるいはそのアプリケーションプログラムの設計に多大な時間やコストを要するという問題があった。

【0 0 0 9】

本発明は、以上のような問題を鑑みてなされたものであり、ユーザが仕様を拡張可能なプロセッサにおいて、ユーザが定義した組込関数を用いた処理動作と同等の動作を行う命令文を、ユーザが定義した組込関数に応じた機械語に最適化することができるコンパイラ、コンパイル方法、及び該コンパイラを用いたプログラム開発ツールを提供することを目的とする。

【0 0 1 0】

【課題を解決するための手段】

上記課題を解決するために、本発明の一態様によるコンパイラは、入力したソースプログラムからオブジェクトコードを生成するコンパイラであって、ソース

プログラム内に記述された命令について文法規則への適合性を解析し、上記命令の組み合わせが組込関数及び該組込関数の動作内容を定義しているか否かを解析する構文解析部と、構文解析部で解析された組込関数及び該組込関数の動作内容の定義を記憶する組込関数定義記憶部と、構文解析部の処理結果に基づいて、ソースプログラムから機械語を生成するコード生成部と、コード生成部により生成された機械語が組込関数定義記憶部に記憶されている組込関数の動作内容と一致する場合、当該機械語を組込関数の動作内容に応じた機械語に最適化するコード最適化部とを備える。

【 0 0 1 1 】

本発明の一態様によるコンパイル方法は、入力したソースプログラムからオブジェクトコードを生成するコンパイラによるコンパイル方法であって、ソースプログラム内に記述された命令の文法規則への適合性を解析し、上記命令の組み合わせが組込関数及び該組込関数の動作内容を定義しているか否かを構文解析部が解析する段階と、構文解析部により解析された組込関数及び該組込関数の動作内容の定義を組込関数定義記憶部に記憶する段階と、構文解析部の処理結果に基づいて、コード生成部がソースプログラムから機械語を生成する段階と、コード生成部により生成された機械語が組込関数定義記憶部に記憶された組込関数の動作内容と一致する場合、コード最適化部が当該機械語を組込関数の動作内容に応じた機械語に最適化する段階とを含む。

【 0 0 1 2 】

本発明の一態様によるプログラム開発ツールは、ユーザ定義のハードウェアを導入したプロセッサのためのアプリケーションプログラムを設計するためのプログラム開発ツールであって、(a)アプリケーションプログラム内に記述された命令をトークンに分割する字句解析部と、トークンについて文法規則への適合性を解析し、ユーザ定義命令のハードウェア定義を入力して組込関数及び該組込関数の動作内容の定義に変換する構文解析部と、構文解析部で変換された組込関数及び該組込関数の動作内容の定義を記憶する組込関数定義記憶部と、字句解析部及び構文解析部の処理結果に基づいて、アプリケーションプログラムから機械語を生成するコード生成部と、コード生成部により生成された機械語が組込関数定

義記憶部に記憶されている組込関数の動作内容と一致する場合、当該機械語を組込関数の動作内容に応じた機械語に最適化するコード最適化部とを含むコンパイラと、(b) コンパイラによりコンパイルされたアプリケーションプログラムをデバッグするシミュレータとを備える。

【0013】

ユーザがソースプログラムを記述する際に、ユーザ定義の組み込み関数を明示的に呼び出さなくても、ユーザ定義の組込関数を用いた処理動作と同等の処理動作を行う命令文を、その組込関数に応じた機械語に最適化することができる。

【0014】

【発明の実施の形態】

以下、本発明の実施形態を図面に基づいて説明する。尚、各図面を通じて同一もしくは同等の部位や構成要素には、同一もしくは同等の参照符号を付し、その説明を省略もしくは簡略化する。

【0015】

図1は、本発明の実施形態におけるコンパイラ10の構成を例示する概略ブロック図である。コンパイラ10は、アプリケーションプログラム開発ツール等のコンピュータシステムに搭載され、プログラミング言語で記述されたソースプログラム1を機械言語に翻訳する翻訳プログラムであり、字句解析部11、構文解析部12、中間コード生成部13、中間コード最適化部14、コード生成部15、コード最適化部16、コード出力部17などを含んでいる。

【0016】

まず、字句解析部11がソースプログラム1を入力して字句解析を行う。字句解析部11は、ソースプログラム1に高級言語で記述された記述された命令文を意味のある最小の単位であるトークン(token)に分割する「字句解析」を行う。「トークン」の代表例は、プログラミング言語のキーワード、演算子、変数名、定数、区切り記号等である。

【0017】

次に、構文解析部12は、字句解析部11の処理結果を入力して構文解析を行う。構文解析部12が行う「構文解析」では、字句解析部11によって変数名や

記号などのトークンに分割された命令について、それぞれのプログラミング言語で定められた文法規則に適合しているか否かがチェックされる。更に、構文解析部 12 は、トークンに分割された命令若しくはこれら命令の組み合わせが、ユーザ定義の組込関数及び該組込関数の動作内容を定義しているか否かを解析し、解析した組込関数及びその動作内容の定義を組込関数定義記憶部である記号表 20 に記憶する。

【0018】

その後、中間コード生成部 13 による中間コードの生成、中間コード最適化部 14 による中間コードの最適化を経て、コード生成部 15 によるコード生成が行われる。コード生成部 15 による「コード生成」では、それまでに行われたソースプログラム 1 の最小単位への分割、構文エラーのチェックなどの結果を受けて、コード・ジェネレータの機能を用いてソースプログラム 1 を、アセンブラ形式や 2 進数形式などの機械語へ変換する処理が行われる。

【0019】

更に、コード最適化部 16 によるコード最適化が行われる。コード最適化部 16 による「コード最適化」では、直前のコード生成部 15 により変換された機械語に、実際の処理効率を向上させるための変更が加えられる。また、コード最適化部 16 は、コード生成部 15 により生成された機械語が、組込関数定義記憶部（記号表 20）に記憶されているユーザが定義した組込関数の動作内容と一致する場合、その機械語をユーザが定義した組込関数の動作内容に応じた機械語に最適化する。

【0020】

そして、コード出力部 17 は、コード最適化部 16 によるコード最適化結果に基づいてオブジェクトコード 2 を生成する。

【0021】

ここで、コンパイラ 10 が中間コード生成部 13 及び中間コード最適化部 14 によって中間コードの生成を行うのは、構文解析の直後にコード生成をすると、生成されるプログラム・サイズが大きくなり、変換処理が効率よく行われない場合があるため、ソースプログラム 1a と等価で、しかも簡潔な言語である中間コ

ードに変換してから翻訳処理を行う。中間コード生成及び中間コード最適化を省略してもオブジェクトコード2は得られる。

【0022】

図2は、中間コードを生成しないでソースプログラム1からオブジェクトコード2を生成するコンパイラ10aの構成を示す概略ブロック図である。図2に示すように、コンパイラ10aは、字句解析部11a、構文解析部12a、コード生成部15a、コード最適化部16a、コード出力部17aを経て、ソースプログラム1からオブジェクトコード2を生成する。構文解析部12aは、字句解析部11aによって分割されたトークンの組み合わせが、ユーザが定義した組込関数及び該組込関数の動作内容を定義しているか否かを解析し、解析した組込関数及びその動作内容の定義を記号表20aに記憶する。コード最適化部16aは、コード生成部15aにより生成された機械語が、記号表20aに記憶されているユーザが定義した組込関数の動作内容と一致する場合、その機械語をユーザが定義した組込関数動作内容に応じた機械語に最適化する。

【0023】

[第1実施例]

プロセッサ・アーキテクチャや命令セットなどの仕様をユーザが拡張可能なプロセッサにおいて、拡張した仕様に応じたハードウェア動作を実現するために、ユーザが独自に組込関数を定義できるようにする。以下、第1の実施例では、プログラミング言語としてC言語をモデルにして説明する。

【0024】

図4(a)及び(b)は、ユーザが拡張した仕様に応じて組込関数を定義した例を示している。また、図5(a)は、図4に示したユーザ定義の組込関数を明示的に呼び出す命令列の記述例を示しており、図5(b)は、図5(a)に示す命令列から生成される機械語列を例示している。図6(a)は、図4に示したユーザ定義の組込関数と同等の処理動作を実行する命令列を例示しており、図6(b)は、図6(a)に示す命令列から生成される最適化前の機械語列、図6(c)は最適化後の機械語列を例示している。

【0025】

図 4 (a) の組込関数定義 F 1 1 では、予約語“__asm”, “__reg_src”を用いて、組込関数“uci”を宣言し仮引数等の属性を定義すると共に、この組込関数“uci”の動作内容 P 1 1 をプログラミング言語で定義する。図 4 (b) の組込関数定義 F 1 2 は、図 4 (a) の組込関数定義 F 1 1 を再定義したものであり、図 4 (a) の動作内容 P 1 1 と異なる記述で動作内容 P 1 2 を定義している。つまり、図 4 (a) ~ (b) に示す組込関数はその動作内容の記述方法が異なるだけで、結果的に同等の動作内容を有している。

【 0 0 2 6 】

このように、1つのユーザ定義の組込関数を何度でも（何種類でも）再定義可能とし、再定義された定義の数が増えれば増えるほど、コンパイラ 1 0 のコード最適化部 1 6 が最適化可能と判定できる命令列の特定を容易にすることができる。

【 0 0 2 7 】

コンパイラ 1 0 は、図 4 に示したようなユーザ定義の組込関数の記述に対して、オブジェクトコード 2 を生成しない。従って、このような組込関数定義をソースプログラム 1 のヘッダファイルに記述しても良いし、あるいは図 3 に示すように組込関数定義をソースプログラム 1 とは別のファイル（組込関数情報ファイル 3）に格納しておき、コンパイラ 1 0 は組込関数情報ファイル 3 からユーザ定義の組込関数定義を読み込んでコンパイルしてもよい。

【 0 0 2 8 】

ユーザ定義の組込関数の定義は、わずかな予約語（“__asm”, “__reg_src”など）を追加しただけであるため、組込関数定義をソースプログラム 1 中に記述するようにすると、ユーザは組込関数の内容を理解でき、組込関数の動作定義から最適化が容易なプログラミングが容易となる。更に、ソースプログラム 1 の管理も容易となる。

【 0 0 2 9 】

図 7 はコンパイラ 1 0 の構文解析部 1 2 の詳細な処理動作を例示するフローチャートである。

【 0 0 3 0 】

構文解析部 1 2 は、字句解析部 1 1 の処理結果を入力し、入力した各トークンについて、図 7 に示すような解析処理を実行する。構文解析部 1 2 はステップ S 0 1 において、入力したトークンが関数宣言であるか否かを判定する。判定の結果、関数宣言でない場合、ステップ S 0 2 へ進み、構文解析部 1 2 は通常の構文解析処理を実行する。

【 0 0 3 1 】

ステップ S 0 1 の判定の結果、関数宣言である場合、構文解析部 1 2 はステップ S 0 3 において、関数宣言に独自の予約語 “_asm” が付加されているか否かを判定する。判定の結果、予約語 “_asm” が付加されていなければ、構文解析部 1 2 はユーザが拡張定義した組込関数ではないと判断し、ステップ S 0 4 の通常の関数宣言処理へ移行する。

【 0 0 3 2 】

ステップ S 0 3 の判定の結果、予約語 “_asm” が付加されている場合、構文解析部 1 2 は、ユーザが定義した組込関数であると判断し、ステップ S 0 5 において、その組込関数の宣言がプロトタイプ宣言なのか、関数定義なのかを判定する。ここで、「プロトタイプ宣言」とはユーザ定義の組込関数のうち、仮引数の型情報や識別子の名前の定義のみで、動作内容の定義を持たない組込関数の宣言をいう。

【 0 0 3 3 】

ステップ S 0 5 の判定の結果、そのユーザ定義の組込関数の宣言がプロトタイプ宣言である場合、構文解析部 1 2 はステップ S 0 6 において、組込関数の仮引数の型情報や識別子の名前の解釈を行い、仮引数の型情報や識別子の名前の指定方法に誤りがあるか否かを判定する。判定の結果、仮引数の型情報や識別子の名前の指定方法に誤りがなければ、構文解析部 1 2 はステップ S 0 7 においてユーザ定義の組込関数の定義内容を記号表 2 0 へ登録する。逆に、仮引数の型情報や識別子の名前の指定方法に誤りがあれば、構文解析部 1 2 はステップ S 0 8 においてエラーメッセージを出力する。

【 0 0 3 4 】

一方、ステップ S 0 5 の判定の結果、ユーザ定義の組込関数の宣言がプロトタ

イブ宣言でない場合、構文解析部 1 2 はステップ S 1 0 において、組込関数の仮引数の型情報や識別子の名前の解釈を行い、仮引数の型情報や識別子の名前の指定方法に誤りがあるか否か、組込関数の動作内容の定義に文法的な誤りがあるか否かを判定する。判定の結果、仮引数の型情報や識別子の名前の指定方法や動作内容の定義に誤りがある場合、ステップ S 0 8 においてエラーメッセージを出力する。

【 0 0 3 5 】

ステップ S 1 0 の判定の結果、仮引数の型情報や識別子の名前の指定方法や動作内容の定義に誤りがない場合、ステップ S 1 1 において、構文解析部 1 2 は、このユーザ定義の組込関数が、既に記号表 2 0 に定義済みの組込関数の再定義であるか否かを判定する。判定の結果、既に定義済みの組込関数の再定義でない場合、ステップ S 1 2 において、構文解析部 1 2 は、ユーザ定義の組込関数及び定義内容の定義を記号表 2 0 へ登録する。

【 0 0 3 6 】

ステップ S 1 1 の判定の結果、このユーザ定義の組込関数が既に記号表 2 0 に定義済みの組込関数の再定義である場合、構文解析部 1 2 はステップ S 1 3 において、この組込関数と定義済みの組込関数の動作定義が実質的に一致しているか否かを判定し、例えば、図 4 (a) の関数定義と図 4 (b) の関数定義のように動作定義が実質的に一致している場合は、ステップ S 1 3 において、組込関数及び定義内容の定義を記号表 2 0 へ登録する。逆に、一致していない場合は、ステップ S 1 4 においてエラーメッセージを出力する。尚、ステップ S 1 1 における再定義された関数の動作内容と実質的に同じであるか否かの判定処理は、一般的なコンパイラで用いられている「共通式の検出」技術により実現できる。

【 0 0 3 7 】

図 4 (a) 及び図 4 (b) に示した組込関数定義に対して、以上の構文解析部 1 2 による処理を実行すると、結果として図 8 に例示するようなデータが記号表 2 0 に登録される。

【 0 0 3 8 】

図 8 に示すように、記号表 2 0 のハッシュテーブル 2 1 内のハッシュ値が指し

示す先に組込関数の定義テーブル 2 2 1, 2 2 2, …が格納されている。各定義テーブル 2 2 1, 2 2 2, …は、ハッシュ値、先頭の機械語、組込関数名、引数、引数の種類、動作定義内容 2 3 1, 2 2 2, …、最適化結果などが格納されている。動作定義内容 2 3 1, 2 2 2, …には、動作内容を示す機械語列が動作順序に並べられて格納されており、構文解析部 1 2 及びコード最適化部 1 6 におけるユーザ定義の組込関数の機械語への変換処理を実現している。

【 0 0 3 9 】

図 9 は、コード最適化部 1 6 が実行する処理の手順例を示すフローチャートである。

【 0 0 4 0 】

まず、ステップ S 2 1 において、コード最適化部 1 6 はコード生成部 1 5 により生成された機械語が、ユーザが定義した組込関数の明示的呼び出しであるかを判定する。判定の結果、図 5 (a) に例示するような組込関数の明示的呼び出しである場合、コード最適化部 1 6 は、図 5 (b) に例示するようなユーザが定義した組込関数“uci”の動作内容に応じた機械語に最適化する。

【 0 0 4 1 】

ステップ S 2 1 の判定の結果、図 6 (a) に例示するような組込関数の明示的呼び出しでない場合、コード最適化部 1 6 はステップ S 2 2 において、図 6 (a) の命令文 T 2 1, T 2 2 にそれぞれ対応する機械語 M 2 1, M 2 2 が、記号表 2 0 に登録されている組込関数の定義内容とそれぞれ一致するかどうか検査する。機械語列 M 2 1 について具体的に説明すると、先頭の機械語列“add \$3,\$1,10”に対して、ハッシュ検索を行うことにより、図 8 の記号表 2 0 に登録された組込関数“uci”の第一の定義 (#num1) に対応する定義内容に一致し、次に、機械語列“add \$4,\$0,10”に対して、ハッシュ検索を行うことにより、同じく組込関数“uci”の第一の定義 (#num1) に対応する定義内容に一致する。従って、機械語列 M 2 1 は、ステップ S 2 3 において、図 6 (c) の機械語列 M 2 3 に最適化され、出力される。同様に、機械語列 M 2 2 は、ステップ S 2 3 において、組込関数“uci”の動作内容に応じた、図 6 (c) の機械語列 M 2 4 に最適化されて、出力される。

【 0 0 4 2 】

ステップ S 2 2 の検査の結果、機械語の組み合わせが記号表 2 0 に登録されている組込関数の定義内容と一致しない場合は、通常の機械語であると判断し、通常の機械語を生成する。図 6 に示した例であれば、図 6 (b) の機械語列 M 2 1 , M 2 2 がそのまま生成される。

【 0 0 4 3 】

以上説明したコンパイラ 1 0 による処理結果として、入力ソースプログラム 1 の例を図 1 0 に示し、生成されるオブジェクトコード 2 (アセンブラ) の例を図 1 1 に示す。

【 0 0 4 4 】

以上説明したように、第 1 実施例によれば、ユーザがソースプログラム 1 を記述する際に、ユーザ定義の組み込み関数を明示的に呼び出さなくても、ユーザが定義した組込関数を用いた処理動作と同等の処理動作を行う命令文を、その組込関数に応じた機械語に最適化することができる。

【 0 0 4 5 】

更に、ユーザ定義の組込関数の定義を、何度でも (何種類でも) 再定義可能とすることで、ユーザ定義の組込関数の機械語への変換において変換候補となる命令文が増え、より効果的な最適化が実現できる。

【 0 0 4 6 】

[第 2 実施例]

第 1 実施例では、C 言語によってユーザ定義の組込関数を記述する例を示したが、ユーザ定義の組込関数を記述するプログラミング言語は C 言語に限らない。

【 0 0 4 7 】

そこで、第 2 実施例では、ハードウェア記述言語 (例えば、VerilogHDL) で記述されたユーザ定義の組込関数を入力して、記号表 2 0 を作成・登録し、ソースプログラム 1 中の同等の処理動作を行う命令文をユーザ定義の組込関数に応じた機械語に最適化するコンパイラ 1 0 の例を示す。

【 0 0 4 8 】

図 1 2 は、VerilogHDL で記述されたユーザ定義の組込関数 (命令動作定義) の

例（命令動作定義ファイル名は"add10_and_1.v"）を示しており、動作内容の命令語列 P 4 1 を含んだ記述となっている。同様に、図 1 3 は、VerilogHDLで記述された命令動作定義の例（命令動作定義ファイル名は"add10_and_2.v"）を示しており、動作内容の命令語列 P 4 2 を含んでいる。

【 0 0 4 9 】

尚、図 1 2 における動作内容の命令語列 P 4 1 は、第 1 実施例の図 1 0 における動作内容の命令語列 P 1 1 と同等の処理を実現し、図 1 3 における動作内容の命令語列 P 4 2 は、図 1 0 における動作内容の命令語列 P 1 2 と同等の処理を実現する。つまり、第 1 実施例と同様に、VerilogHDLで記述された命令動作定義の再定義が可能である。

【 0 0 5 0 】

図 1 4 (a) は、コンパイラ 1 0 が入力するソースプログラム 1 の例を示している。ソースプログラム 1 中には、VerilogHDLで記述された命令動作定義ファイルの宣言 H 4 1、H 4 2 と、ユーザが定義した組込関数の明示的呼び出しを行う命令文 T 4 1、命令文 T 4 1 と同等の処理を実現する命令文 T 4 2 が C 言語で記述されている。このようなソースプログラム 1 を入力したコンパイラ 1 0 の構文解析部 1 2 の処理動作例を図 1 5 に例示する。

【 0 0 5 1 】

図 1 5 に示すように、構文解析部 1 2 は字句解析部 1 1 の処理結果を入力し、ステップ S 3 1 において、入力したトークンがVerilogHDLの命令動作定義ファイルの宣言（"#pragma input HDL"）であるか否かを判定する。判定の結果、VerilogHDLの命令動作定義ファイルの宣言でない場合、構文解析部 1 2 はステップ S 3 2 において通常の構文解析処理を実行する。

【 0 0 5 2 】

ステップ S 3 1 の判定の結果、VerilogHDLの命令動作定義ファイルの宣言である場合、構文解析部 1 2 はステップ S 3 3 において、VerilogHDLの命令動作定義ファイルを入力する。そして、構文解析部 1 2 はステップ S 3 4 において、入力した命令動作定義ファイルに記述されている命令動作定義に、文法的な誤りがあるか否かを判定する。

【 0 0 5 3 】

ステップ S 3 4 の判定の結果、命令動作定義に誤りがあれば、構文解析部 1 2 はステップ S 0 8 においてエラーメッセージを出力し、命令動作定義に誤りがなければ、構文解析部 1 2 はステップ S 3 6 において、VerilogHDL の命令動作定義の記述をユーザ定義の組み込み関数定義に記述に変換する。例えば、図 1 1 に示した命令動作定義は、図 1 0 における関数定義 F 1 1 に変換され、図 1 2 に示した命令動作定義は、図 1 0 における関数定義 F 1 2 に変換される。

【 0 0 5 4 】

ステップ S 3 6 の変換処理が正常に行われなかった場合、構文解析部 1 2 はステップ S 3 8 においてエラーメッセージを出力する。

【 0 0 5 5 】

逆に、ステップ S 3 6 の変換処理が正常に行われた場合、構文解析部 1 2 はステップ S 4 1 において、変換された組込関数が記号表 2 0 に既に定義済みの組込関数の再定義であるか否かを判定する。判定の結果、この組込関数が既に定義済みの組込関数の再定義でない場合、ステップ S 4 2 において、構文解析部 1 2 はこの組込関数及び定義内容の定義を記号表 2 0 へ登録する。

【 0 0 5 6 】

ステップ S 4 1 の判定の結果、この組込関数が既に記号表 2 0 に定義済みの組込関数の再定義である場合、構文解析部 1 2 はステップ S 4 3 において、この組込関数と定義済みの組込関数の動作定義が実質的に一致しているか否かを判定し、実質的に一致している場合は、ステップ S 1 3 において、この組込関数及び定義内容の定義を記号表 2 0 へ登録する。逆に、一致していない場合は、ステップ S 1 4 においてエラーメッセージを出力する。

【 0 0 5 7 】

以上の構文解析部 1 2 による処理を実行した結果、図 1 6 に例示するようなデータが記号表 2 0 に登録される。結果的に、図 8 に示した記号表 2 0 とほぼ同等な内容になっている。

【 0 0 5 8 】

尚、第 2 実施例におけるコンパイラ 1 0 の構成要素のうち、構文解析部 1 2 以

外の各構成要素は、図 1 に示したコンパイラ 1 0 の各構成要素と同様である。

【0059】

以上説明したコンパイラ 1 0 による処理結果として、図 1 4 (a) に示した入力のソースプログラム 1 が、図 1 4 (b) に示すオブジェクトコード 2 (アセンブラ) に最適化されて生成される。

【0060】

このように、第 2 実施例によれば、ハードウェア記述言語によって記述されたユーザ定義の組込関数と同等の処理動作を行う命令文を、ユーザの拡張定義の組込関数に応じた機械語に最適化することができる。

【0061】

また、ユーザは、プログラミングの際にハードウェア記述言語によって記述されたユーザ定義の組込関数の定義を読むことで、ハードウェアの実装方法が容易に理解できるため、ユーザが拡張した仕様に基づいたプログラムの動作時の最適化を図ることができる。

【0062】

尚、図 1 5 に示したHDLファイルの入力処理 (ステップ S 3 3) の後で、字句解析や構文解析の処理 (ステップ S 3 4) を行う例を示したが、ソースプログラム 1 中にVerilogHDLの記述を混在可能とするために、字句解析部 1 1 や構文解析部 1 2 にVerilogHDLの字句解析や構文解析処理機能を持たせるようにしてもよい。

【0063】

〔第 3 実施例〕

第 3 実施例では、第 2 実施例で説明したコンパイラ 1 0 を用いたプログラム開発ツールについて詳細に説明する。このプログラム開発ツールは、例えば、ユーザ定義のハードウェアを導入したプロセッサと、そのプロセッサのためのアプリケーションプログラムを設計するための開発ツールとして用いることができる。

【0064】

初めに、比較例として、特許文献 1 に記載のコンパイラを利用したプログラム開発ツールの処理動作例を図 1 7 に示す。

【 0 0 6 5 】

図 1 7 に示すように、開発対象のアプリケーションプログラムを作成し（ステップ S 5 1）、作成した開発対象プログラムのユーザ定義命令の該当個所を、ユーザ定義命令を記述した組込関数へ置き換える修正を施す（ステップ S 5 2）。そして、修正した開発対象のアプリケーションプログラムを特許文献 1 に記載のコンパイラによりコンパイルし（ステップ S 5 3 a）、シミュレーションによるデバッグを行う（ステップ S 5 3 b）。

【 0 0 6 6 】

ステップ S 5 3 のシミュレーション結果が“OK”でない場合、再度ステップ S 5 2 へ戻って、ユーザ定義命令の組込関数への置き換えからやり直す。

【 0 0 6 7 】

ステップ S 5 3 のシミュレーション結果が“OK”の場合は、修正した開発対象のアプリケーションプログラムから生成された実行形式のオブジェクトコード 2 が得られる（ステップ S 5 5）。また、他方で、出来上がった開発対象のアプリケーションプログラムに基づいて、ユーザ定義命令のハードウェア定義の記述を行い（ステップ S 5 6）、シミュレーションによるデバッグを行う（ステップ S 5 7）。

【 0 0 6 8 】

ステップ S 5 7 のシミュレーション結果が“OK”でない場合、再度ステップ S 5 2 へ戻って、ユーザ定義命令の組込関数への置き換えからやり直すか、ステップ S 5 6 へ戻って、ユーザ定義命令のハードウェア定義の記述からやり直す。

【 0 0 6 9 】

ステップ S 5 6 のシミュレーション結果が“OK”の場合、出来上がったユーザ定義命令のハードウェア定義が得られる（ステップ S 5 9）。

【 0 0 7 0 】

以上のような比較例のプログラム開発ツールに対し、次に、第 3 実施例におけるプログラム開発ツールについて、図 1 8 のフローチャートに従って説明する。

【 0 0 7 1 】

図 1 8 に示すように、第 3 実施例におけるプログラム開発ツールでは、例えば

図14(a)に示したような開発対象のアプリケーションプログラムを作成する(ステップS71)。一方で、図12及び図13に示したようなユーザ定義命令のハードウェア定義の記述を行う(ステップS72)。

【0072】

そして、第2実施例で説明したコンパイラ10を用いて、ステップS71で作成した開発対象のアプリケーションプログラムをコンパイルする(ステップS73a)。具体的には、構文解析部12がユーザ定義命令のハードウェア定義を入力して組込関数の定義内容に変換し、変換した組込関数及び該組込関数の動作内容の定義を記号表20に記憶する。そして、コード最適化部16がコード生成部15により生成された機械語が記号表20に記憶された組込関数の動作内容と一致するか否かを判定し、一致する場合、当該機械語を組込関数の定義内容に応じた機械語に最適化する。

【0073】

このようにしてコンパイルされた開発対象のアプリケーションプログラムについて、シミュレーションによるデバッグを行う(ステップS73b)。

【0074】

ステップS73のシミュレーション結果が"OK"でない場合、再度ステップS72へ戻って、ユーザ定義命令のハードウェア定義の記述からやり直す。

【0075】

ステップS73のシミュレーション結果が"OK"である場合、出来上がった開発対象のアプリケーションプログラムから生成された実行形式のオブジェクトコード2が得られ(ステップS76)、ユーザ定義命令のハードウェア定義も得られる(ステップS75)。

【0076】

このように、第3実施例におけるプログラム開発ツールによれば、比較例に示したプログラム開発ツールに比べて、極めて簡単な処理手順で、ユーザ定義のハードウェアを導入したプロセッサと、そのプロセッサのためのアプリケーションプログラムを設計することができる。

【0077】

ユーザは、ソースプログラム1をプログラミングする時に、ユーザ定義のハードウェアを意識しないでプログラミングすることができる。更に、ユーザ定義のハードウェアをプロセッサの仕様に導入した後、ユーザが定義した部分に該当するプログラム記述を、組込関数に置き換える必要がないため、置き換えに伴うミスなどを防止することができ、しかも作業時間を短縮することができる。

【0078】

以上、第1実施例～第3実施例について詳細に説明したが、本発明は、その精神または主要な特徴から逸脱することなく、他の色々な形で実施することができる。

【0079】

例えば、前述の各実施例では、C言語を用いて記述されたソースプログラム1をコンパイルするコンパイラ10を例示したが、プログラミング言語はこれに限定されず、例えばC++言語によって記述されたソースプログラムも同様にしてコンパイルできる。

【0080】

また、前述の各実施例では、予約語“__asm”や予約語“__reg_src”を用いてユーザ定義の組込関数を定義する例を示したが、使用するプログラミング言語に規定されている予約語以外の語句であれば、他の語句を用いてユーザ定義の組込関数を定義することができる。

【0081】

このように、前述の各実施例はあらゆる点で単なる例示に過ぎず、限定的に解釈してはならない。本発明の範囲は、特許請求の範囲によって示すものであって、明細書本文には何ら拘束されない。更に、特許請求の範囲の均等範囲に属する変形や変更は、全て本発明の範囲内のものである。

【0082】

【発明の効果】

本発明によれば、ソースプログラム中でユーザ定義の組み込み関数を明示的に呼び出さなくても、ユーザが定義した組込関数を用いた処理動作と同等の動作を行う命令文を、その組込関数に応じた機械語に最適化できるコンパイラ、コンパ

イル方法及びプログラム開発ツールを提供することができる。

【図面の簡単な説明】

【図 1】

本実施形態におけるコンパイラの構成を例示する概略ブロック図である。

【図 2】

中間コードを生成しないでコンパイルを実行するコンパイラの構成を例示する概略ブロック図である。

【図 3】

組込関数定義を組込関数情報ファイル 3 から入力してコンパイルを実行するコンパイラを例示する概略図である。

【図 4】

C 言語を用いてユーザ定義の組込関数を記述した例を示したイメージ図であり、(a) は第 1 の定義例、(b) は再定義した第 2 の定義例を示している。

【図 5】

図 5 (a) は、図 4 に示したユーザ定義の組込関数を明示的に呼び出す命令列の記述例を示したイメージ図であり、図 5 (b) は、図 5 (a) に示す命令列から生成される機械語列を例示したイメージ図である。

【図 6】

図 6 (a) は、図 4 に示したユーザ定義の組込関数と同等の処理動作を実行する命令列を例示したイメージ図であり、図 6 (b) は、図 6 (a) に示す命令列から生成される最適化前の機械語列、図 6 (c) は最適化後の機械語列を例示したイメージ図である。

【図 7】

第 1 実施例のコンパイラにおける構文解析部による処理手順例を示すフローチャートである。

【図 8】

図 8 に示した構文解析部の処理により作成される記号表の一例を示す図である。

【図 9】

第 1 実施例のコンパイラにおけるコード最適化部による処理手順例を示すフローチャートである。

【図 1 0】

第 1 実施例のコンパイラがコンパイルするソースプログラムの記述例を示すイメージ図である。

【図 1 1】

図 1 0 に示すソースプログラムを第 1 実施例のコンパイラがコンパイルした結果生成されるオブジェクトコードの例を示すイメージ図である。

【図 1 2】

ハードウェア記述言語 (VerilogHDL) を用いてユーザ定義の組込関数を記述した例を示したイメージ図である。

【図 1 3】

ハードウェア記述言語 (VerilogHDL) を用いてユーザ定義の組込関数を記述した例を示したイメージ図である。

【図 1 4】

(a) は第 2 実施例のコンパイラがコンパイルするソースプログラムの記述例を示し、(b) はコンパイルの結果生成されるオブジェクトコードの例を示すイメージ図である。

【図 1 5】

第 2 実施例のコンパイラにおける構文解析部による処理手順例を示すフローチャートである。

【図 1 6】

図 1 4 に示した構文解析部の処理により作成される記号表の一例を示す図である。

【図 1 7】

比較例におけるプログラム開発ツールの処理手順例を示すフローチャートである。

【図 1 8】

第 3 実施例におけるプログラム開発ツールの処理手順例を示すフローチャート

である。

【図 1 9】

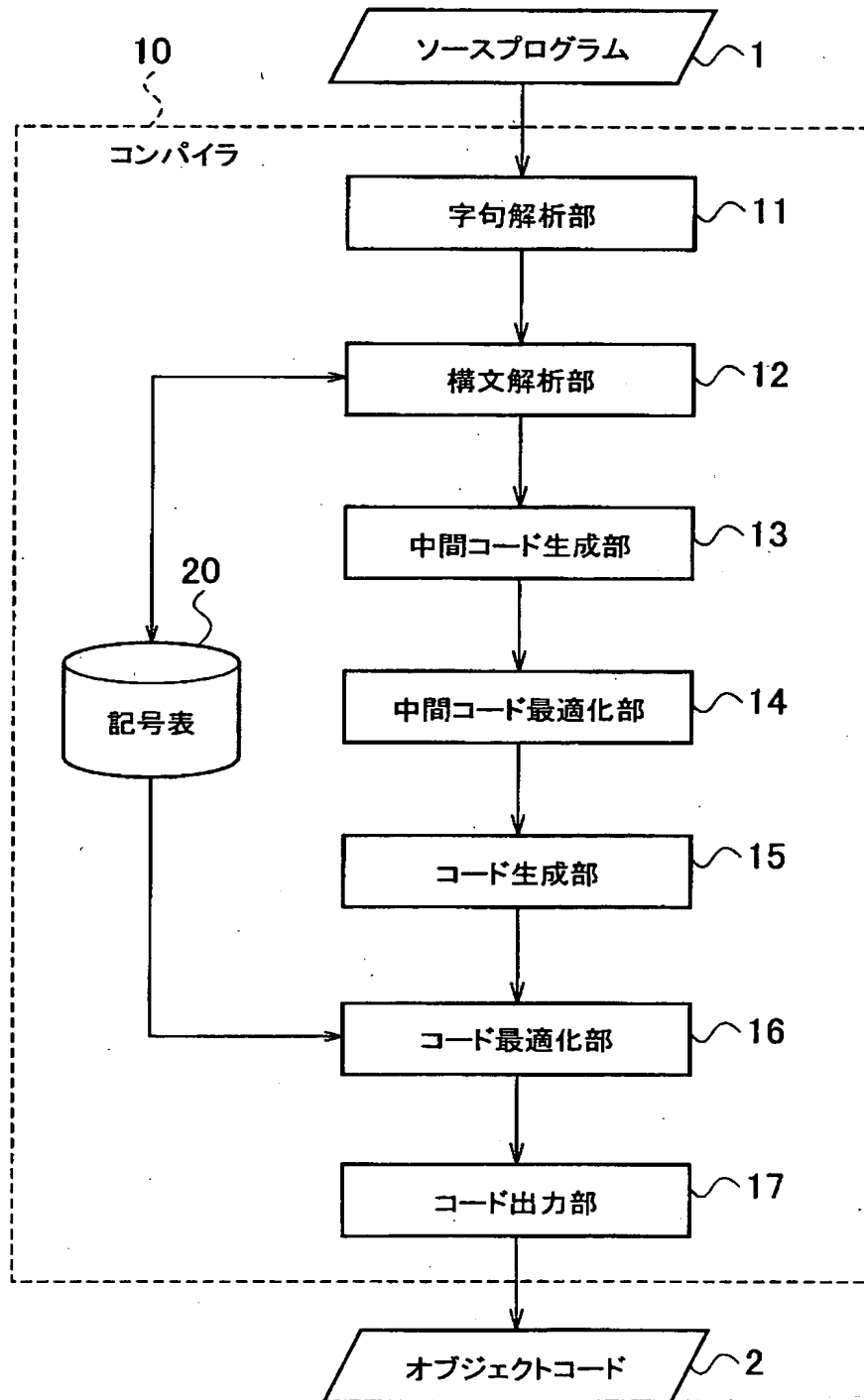
(a) は一般的なソースプログラムの記述例を示し、(b) はコンパイルの結果生成されるオブジェクトコードの例を示すイメージ図である。

【符号の説明】

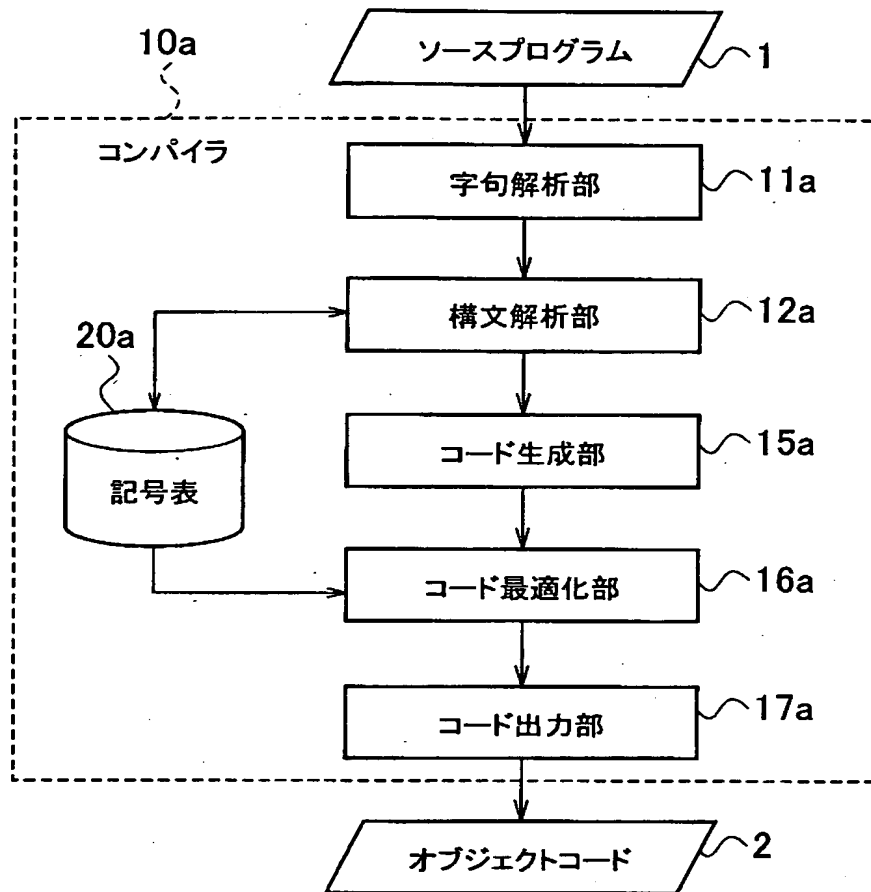
- 1, 1 a … ソースプログラム
- 2 … オブジェクトコード
- 3 … 組込関数情報ファイル
- 1 0, 1 0 a … コンパイラ
- 1 1, 1 1 a … 字句解析部
- 1 2, 1 2 a … 構文解析部
- 1 3 … 中間コード生成部
- 1 4 … 中間コード最適化部
- 1 5, 1 5 a … コード生成部
- 1 6, 1 6 a … コード最適化部
- 1 7, 1 7 a … コード出力部
- 2 0, 2 0 a … 記号表
- 2 1 … ハッシュテーブル
- 2 2 1, 2 2 2 … 定義テーブル
- 2 3 1, 2 2 2 … 動作定義内容

【書類名】 図面

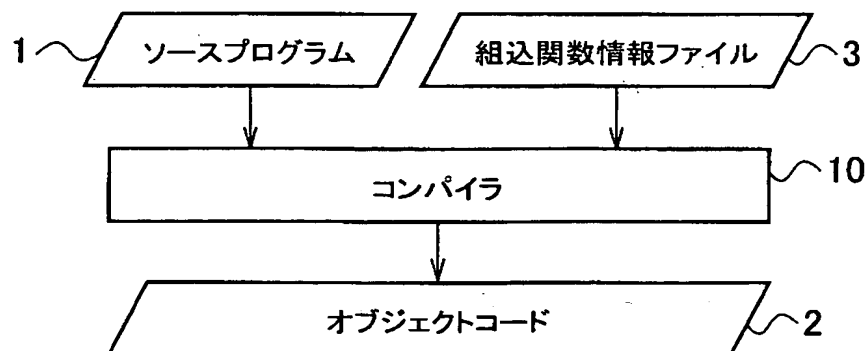
【図 1】



【図 2】

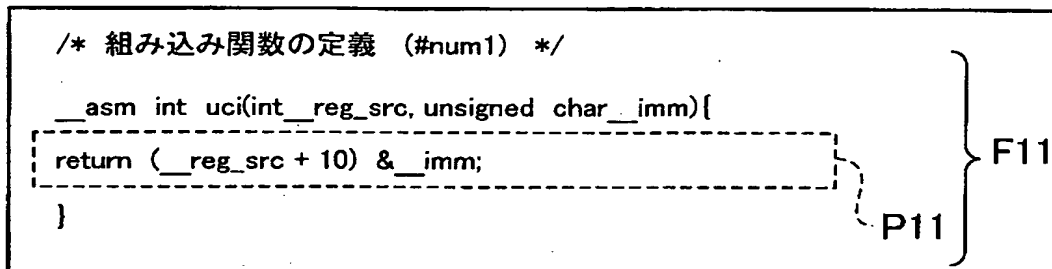


【図 3】

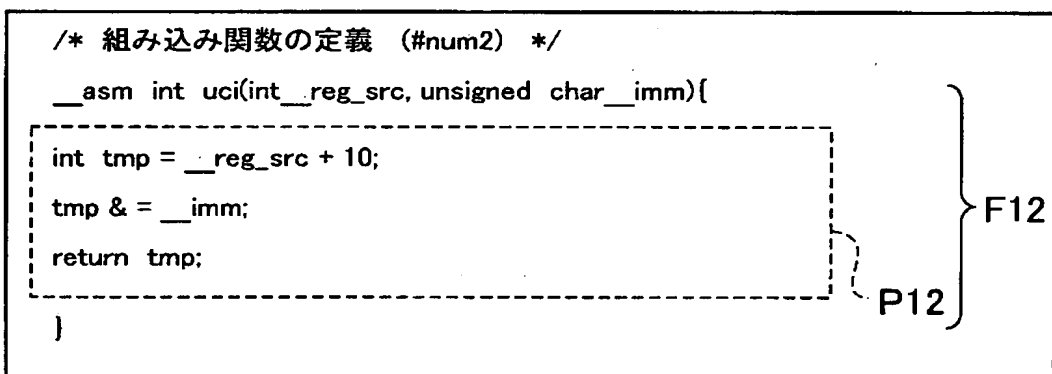


【図4】

(a)



(b)



【図 5】

(a)

```

/* 組み込み関数の明示的呼び出し */
int a, b;
a = uci( b, 255);          . . . . . (T11)
a = uci(a, 127);          . . . . . (T12)
    
```

(b)

```

uci $0, $1, 255          . . . . . (M11)
uci $0, $0, 127          . . . . . (M12)
    
```

【図 6】

(a)

int a, b;	
a = (b+10) &255; (T21)
a = (a+10) &127; (T22)

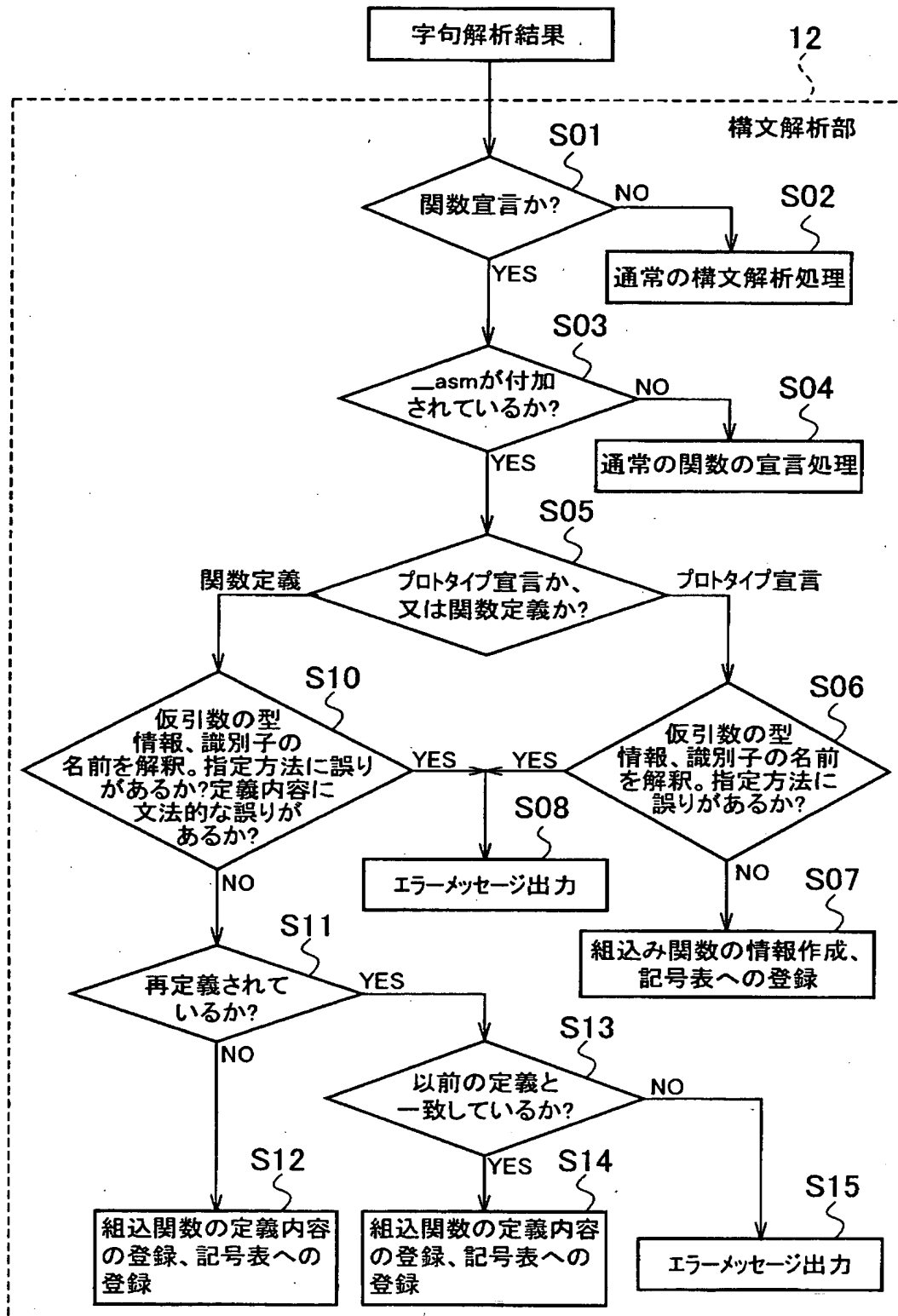
(b)

add \$3, \$1, 10	} (M21)
add \$0, \$3, 255		
add \$4, \$0, 10	} (M22)
add \$0, \$4, 127		

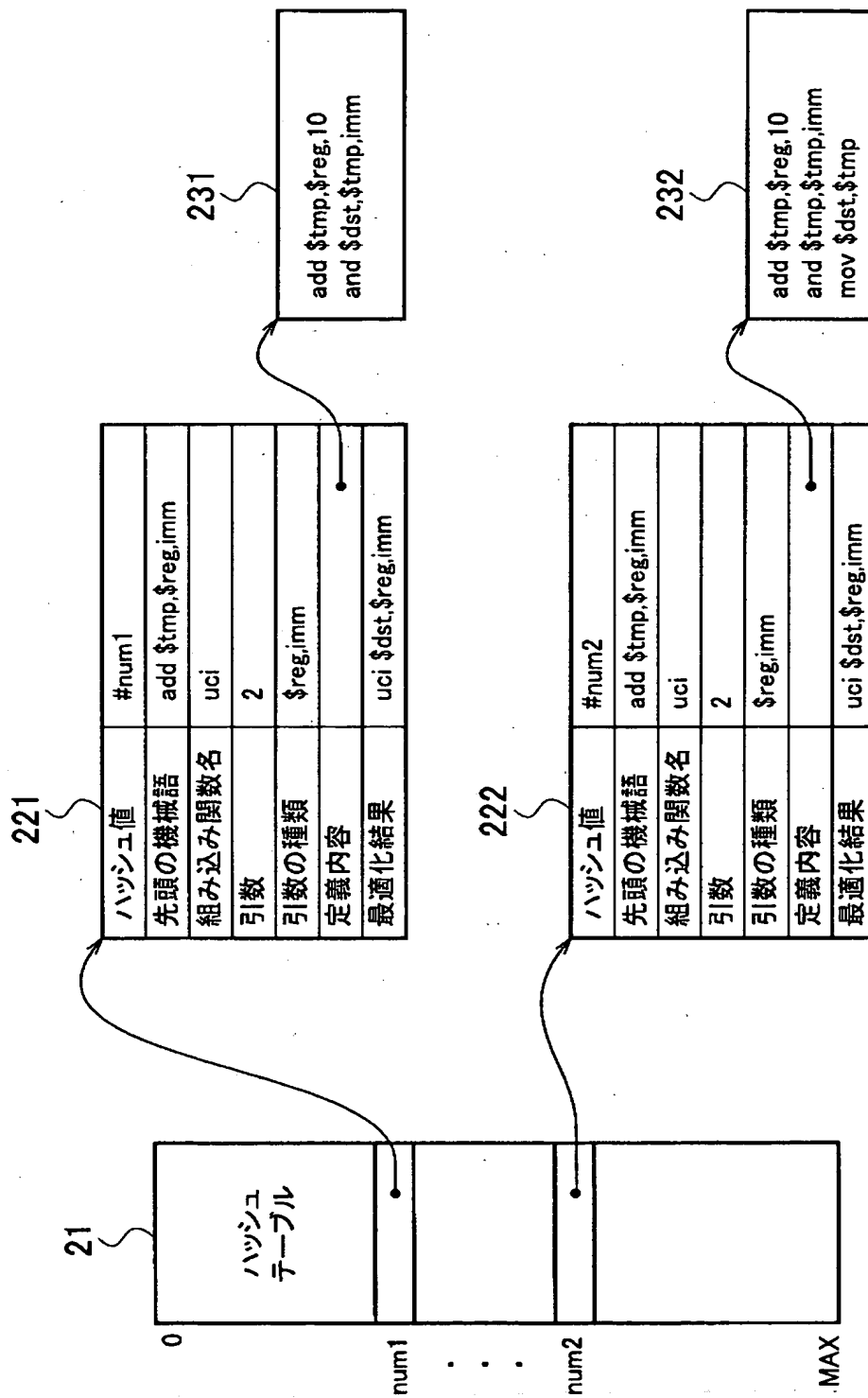
(c)

uci \$0, \$1, 255 (M23)
uci \$0, \$0, 127 (M24)

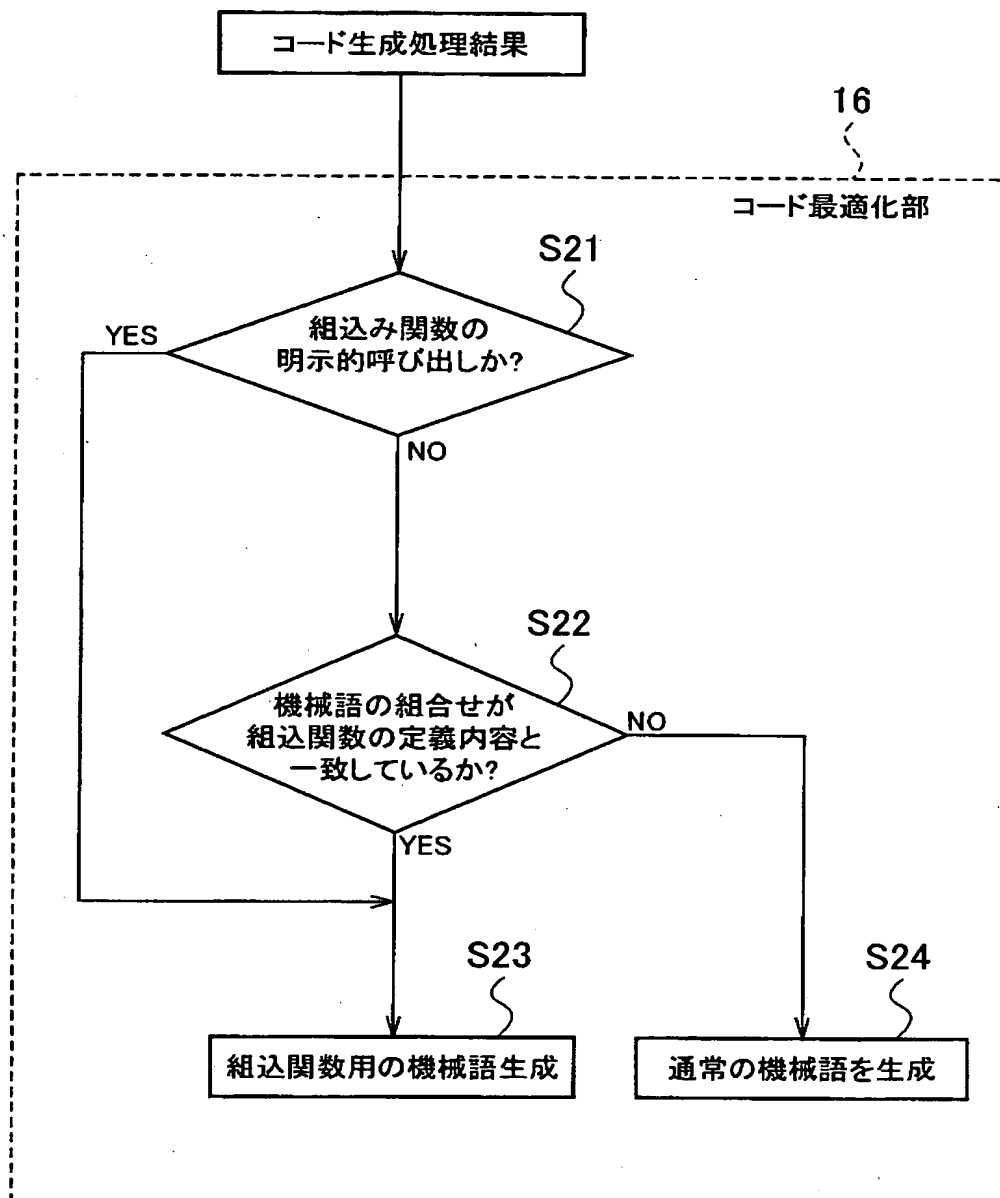
【図 7】



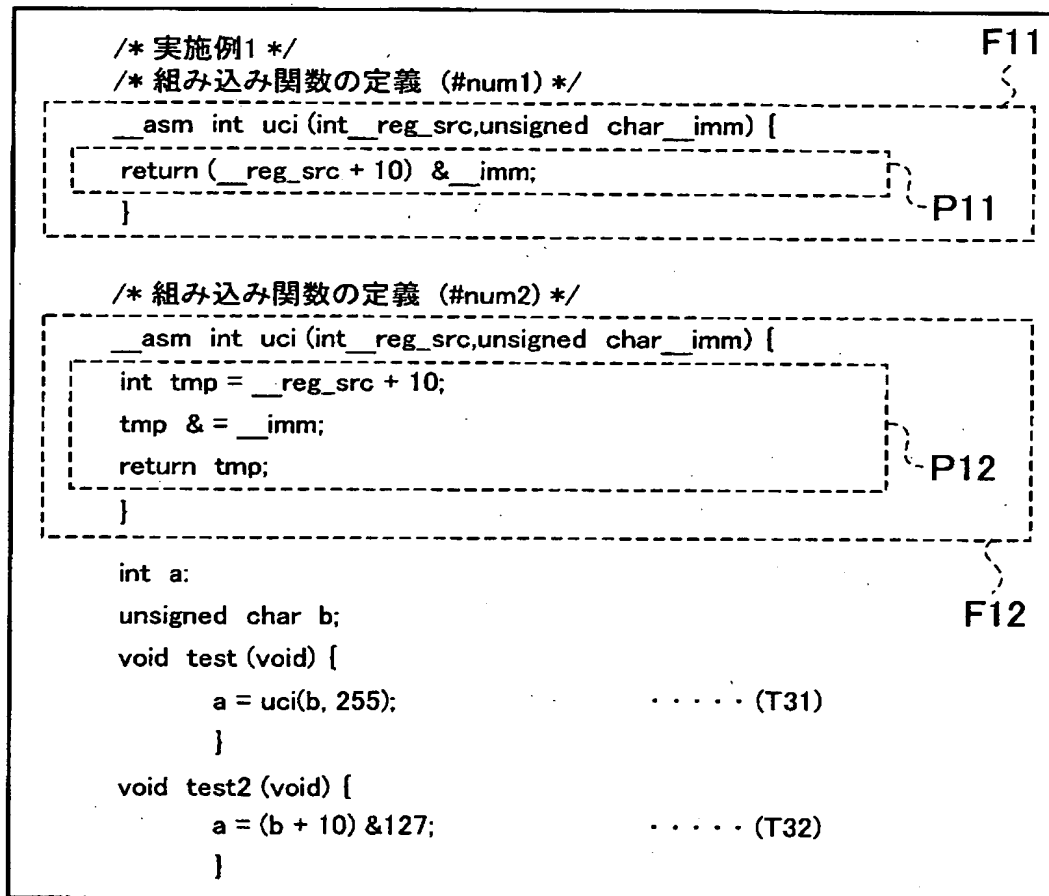
【図 8】



【図9】



【図 1 0】



【図 1 1】

```
__test :  
    lbu    $12, %sdaoff(_b)($14)  
    uci    $11, $12, 255          . . . . . (M31)  
    sw     $11, %sdaoff(_a)($14)  
    ret  
  
__test2 :  
    lbu    $12, %sdaoff(_b)($14)  
    uci    $11, $12, 127         . . . . . (M32)  
    sw     $11, %sdaoff(_a)($14)  
    ret
```

【図 1 2】

```

module uci (
  meucEUCICode,
  meucEUCIRn,
  meucEUCIRm,
  meucEUCIResult
);
input [15:0] meucEUCICode;
input [31:0] meucEUCIRn;
input [31:0] meucEUCIRm;
output [31:0] meucEUCIResult;

```

P41

```

assign meucEUCIResult
  = (meucEUCIRm + 32'h0000000a) & [{ 16 {1'b0}}, meucEUCICode ];

```

```

endmodule

```

【図 1 3】

```
module uci (  
    meucEUCICode,  
    meucEUCIRn  
    meucEUCIRm,  
    meucEUCIResult  
);  
input [15 : 0] meucEUCICode;  
input [31 : 0] meucEUCIRn;  
input [31 : 0] meucEUCIRm;  
output [31 : 0] meucEUCIResult;  
  
wire [31 : 0] tmp;  
wire [31 : 0] imm;
```

P42

```
assign tmp = meucEUCIRm + 32'h0000000a;  
assign imm = {{16 {1'b0 }}, meucEUCICode };  
assign meucEUCIResult = tmp & imm;
```

```
endmodule
```

【図 1 4】

(a)

```

/* 実施例2 */
#pragma input HDL add10_and_1.V      . . . . . (H41)
#pragma input HDL add10_and_2.V      . . . . . (H42)
int a;
unsigned char b;
void test(void){
    a = uci( b, 255);                . . . . . (T41)
}
void test2(void){
    a = (b + 10) & 127;              . . . . . (T42)
}

```

(b)

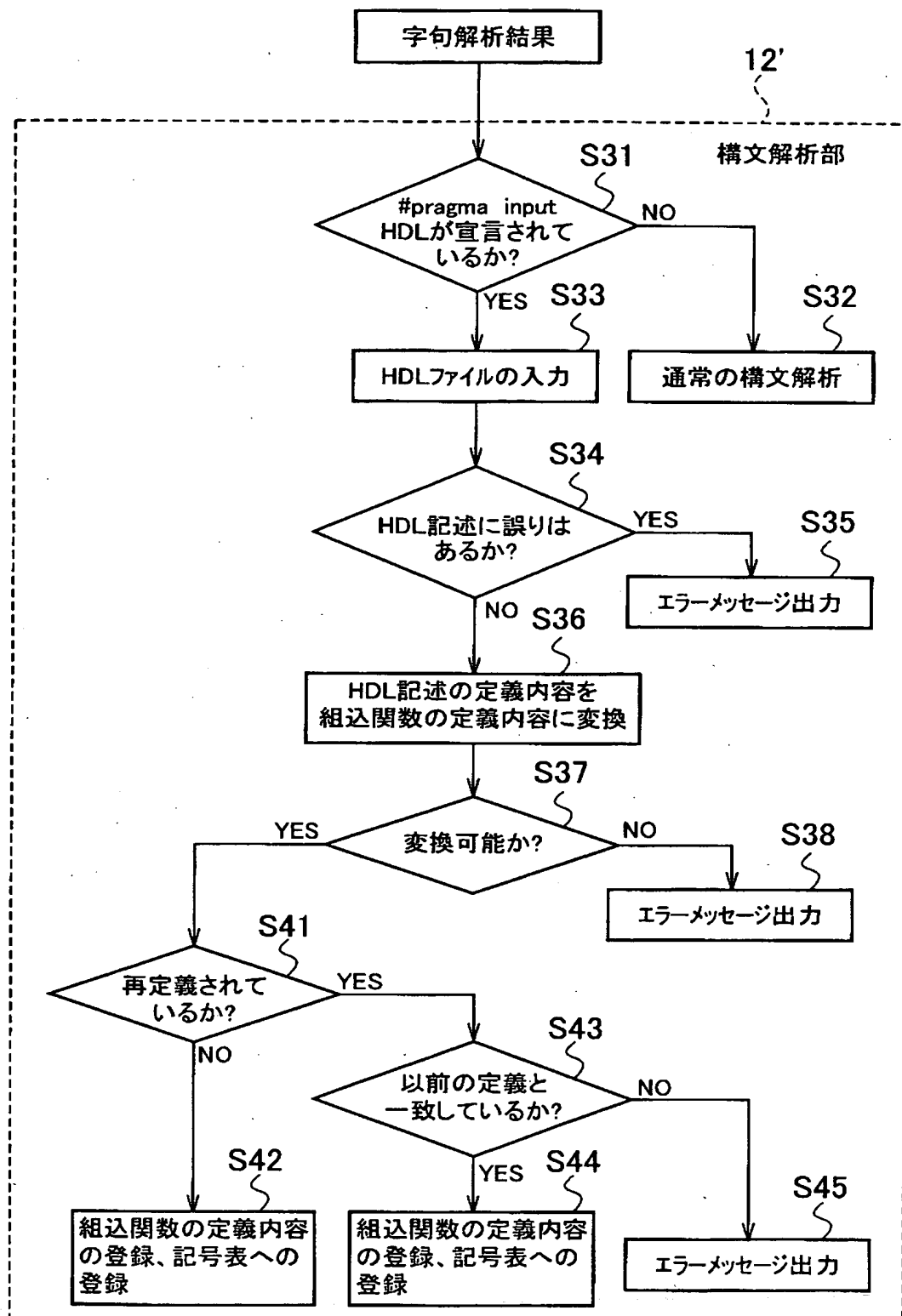
```

_test :
    lbu    $12, %sdaoff(_b)($14)
    uci    $11, $12, 255              . . . . . (M41)
    sw     $11, %sdaoff(_a)($14)
    ret

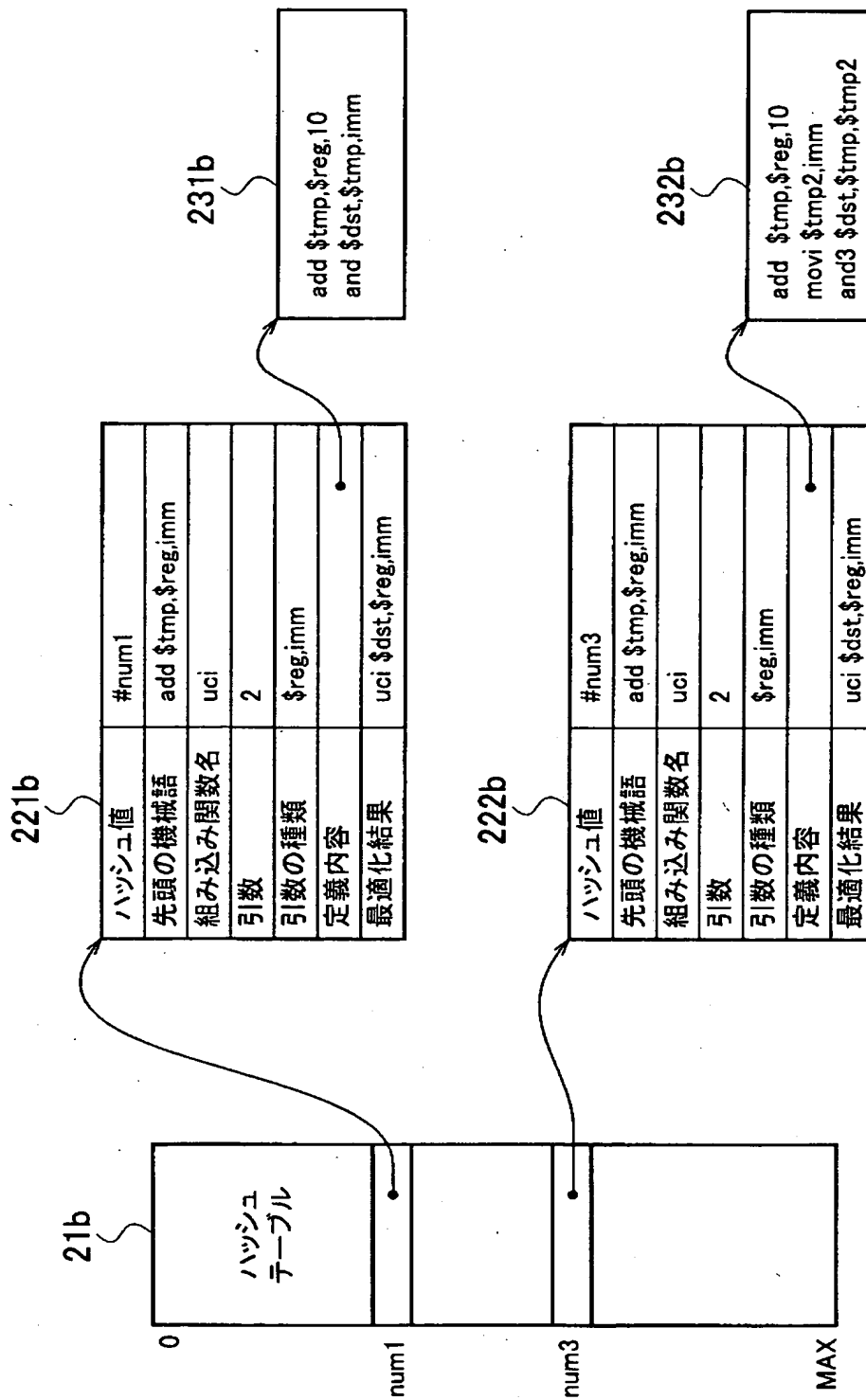
_test2 :
    lbu    $12, %sdaoff(_b)($14)
    uci    $11, $12, 127              . . . . . (M42)
    sw     $11, %sdaoff(_a)($14)
    ret

```

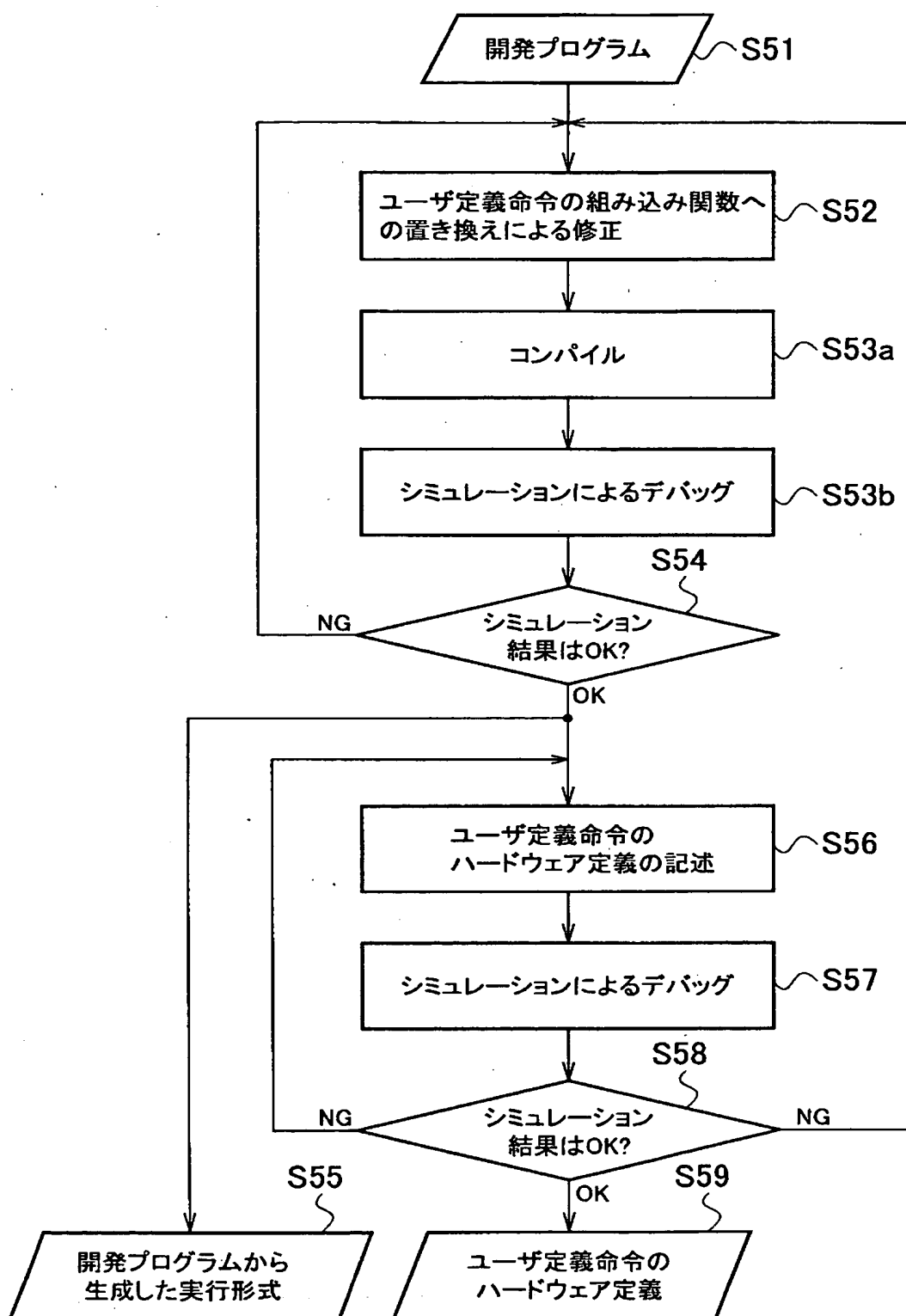
【図15】



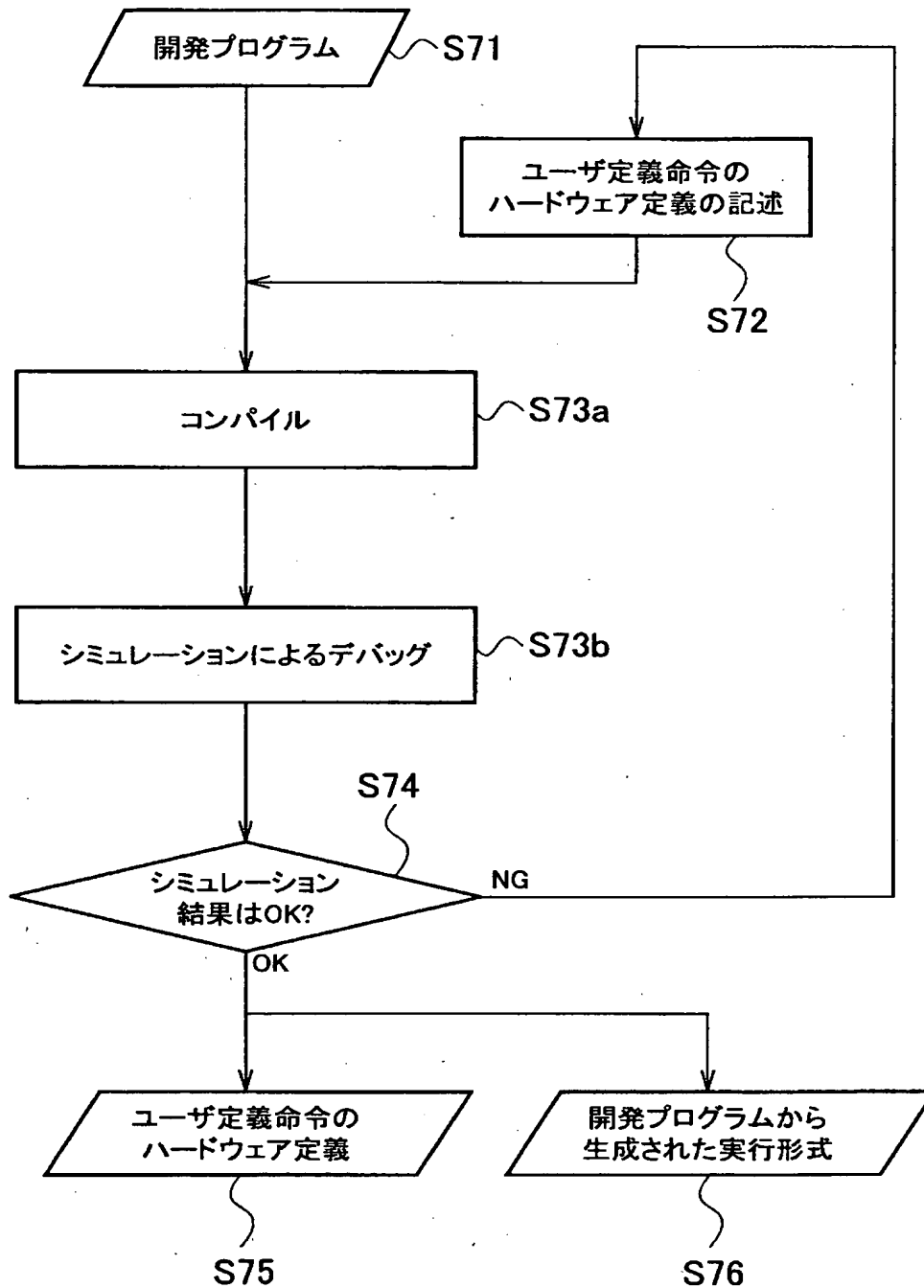
【図 1 6】



【図 17】



【図 18】



【図 1 9】

(a)

<code>__asm int uci(int __reg_src, unsigned char __imm);</code> (F91)
<code>⋮</code>	
<code>a = uci(b, 255);</code> (T91)
<code>⋮</code>	
<code>a = (b+10) & 255;</code> (T92)

(b)

<code>uci \$0, \$1, 255</code> (M91)
<code>⋮</code>	
<code>add \$3, \$1, 10</code>	} (M92)
<code>add \$0, \$3, 255</code>	

【書類名】 要約書

【要約】

【課題】 ユーザが定義した組込関数を用いた処理動作と同等の動作を行う命令文を、ユーザが定義した組込関数に応じた機械語に最適化すること。

【解決手段】 ソースプログラム 1 に記述された命令について文法規則への適合性を解析し、上記命令の組み合わせが組込関数及び該組込関数の動作内容を定義しているか否かを解析する構文解析部 1 2 と、構文解析部 1 2 で解析された組込関数及び該組込関数の動作内容の定義を記憶する記号表 2 0 と、構文解析部 1 2 の処理結果に基づいて、ソースプログラム 1 から機械語を生成するコード生成部 1 5 と、コード生成部 1 5 により生成された機械語が記号表 2 0 に記憶されている組込関数の動作内容と一致する場合、当該機械語を組込関数の動作内容に応じた機械語に最適化するコード最適化部 1 6 とを備えることによる。

【選択図】 図 1

出 願 人 履 歴 情 報

識別番号 [000003078]

1. 変更年月日	2001年 7月 2日
[変更理由]	住所変更
住 所	東京都港区芝浦一丁目1番1号
氏 名	株式会社東芝